Smart Contract Security and Fairness
*A Tale of Two Contending Parties*

Yi Li
Nanyang Technological University

Online
May 13, 2021
Why is blockchain such a big thing?

**Internet** is the information superhighway

**Blockchain** is the Internet of value *(trust)*
Smart Contracts

User-defined computer programs running on top of blockchain

Image curtsey: www.flaticon.com
Smart Contracts

• Managing exchange of digital assets
• Applications across many different sectors
• **Ethereum** in 2020:
  • 825,895 smart contracts created in February
  • 2,855 DApps
  • 31.59K active users / Day
  • 1.143M ($670M) transactions / Day

Sources:
Ethereum Statistics: [https://ycharts.com/indicators/reports/ethereum_statistics](https://ycharts.com/indicators/reports/ethereum_statistics)
In code we trust? No!

Problem: establishing trust between parties with conflicting interests

Smart Contracts

Contract owner

Decentralization
Anonymity

Blockchain

Contract user

Immutability
Transparency

Level of trust
Story 1

Who moved my Ether?
<table>
<thead>
<tr>
<th>Year</th>
<th>Incident</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019/01</td>
<td>51% attack on Ethereum Classic, $200K of Loss</td>
</tr>
<tr>
<td>2018/06</td>
<td>Bithumb Hacks with $31 Million Dollars Stolen</td>
</tr>
<tr>
<td>2018/05</td>
<td><strong>EDU, BAIC Smart Contracts Bugs</strong></td>
</tr>
<tr>
<td>2018/04</td>
<td><strong>BEC, SMT Smart Contracts Bugs</strong></td>
</tr>
<tr>
<td>2018/04</td>
<td>Myetherwallet Suffer from DNS Hijacking</td>
</tr>
<tr>
<td>2018/02</td>
<td>BitGrail Hacks with Stolen Nano Tokens of 170 Million Dollars</td>
</tr>
<tr>
<td>2018/01</td>
<td>Dollar's Coincheck Hacks with 530 Million Dollars</td>
</tr>
<tr>
<td>2017/12</td>
<td>Nicehash Hacks with 4700 BTC Missing with 62 Million Dollars</td>
</tr>
<tr>
<td>2017/06</td>
<td><strong>Bithumb Hacks with 1 Billion Korean Yuan Loss and 30 Thousand User</strong></td>
</tr>
<tr>
<td>2016/08</td>
<td>Info. Leaked Bitfinex Hacks with 120,000 BTC Stolen of 75 Million Dollars</td>
</tr>
<tr>
<td>2016/01</td>
<td><strong>Cryptsy Hacks with 13,000 BTC and 300,000 LTC</strong></td>
</tr>
<tr>
<td>2015/01</td>
<td>Stolen Bitstamp Hacks with 19,000 BTC Stolen</td>
</tr>
<tr>
<td>2014/03</td>
<td>Poloniex Hacks with 12.3% BTC Lost</td>
</tr>
<tr>
<td>2014/02</td>
<td>Mt.Gox Hacks with Followed Bankruptcy</td>
</tr>
</tbody>
</table>
Example: the DAO attack

Attacker’s Contract

```solidity
function moveBalance() {
    dao.withdraw();
}

function() payable {
    dao.withdraw();
}
```

DAO Contract

```solidity
mapping(address =>

function withdraw() {
    uint amount = balances[msg.sender];
    msg.sender.call.value(amount)();
    balances[msg.sender] = 0;
}
```

Calls the default "fallback" function

```
withdraw()
```

```
10 ether
```

```
withdraw()
```

........
Example: the DAO attack

Attacker’s contract

Initiate withdrawal

fallback function

withdraw

overridden by the attacker
triggers another withdraw

DAO contract

withdraw

Ether transfer

-10 ETH

send Ether, pass along gas

update balance

internal state update

Re-entrancy attack with repeated Ether withdrawal

Image source: https://quantstamp.com/blog/what-is-a-re-entrancy-attack
Moral of the story

Contract developers’ expectations ≠ how the contract code actually works
Story 2

All I want is my fair share
An Auction Smart Contract

- Open to all bidders
- **highestBidder** wins the bid
- Latecomer wins when bidding $1 more than the **highestBid**

```solidity
contract CryptoRomeAuction {
  uint256 public highestBid = 0;
  address payable public highestBidder;
  mapping(address=>uint) refunds;
  function bid() public payable{
    uint duration = 1;
    if (msg.value < (highestBid + duration)){
      revert();
    }
    if (highestBid != 0) {
      refunds[highestBidder] += highestBid;
    }
    highestBidder = msg.sender;
    highestBid = msg.value;
  }
}
```
An Auction Smart Contract

- Open to all bidders
- \textit{highestBidder} wins the bid
- Latecomer wins when bidding $1$ more than the \textit{highestBid}

<table>
<thead>
<tr>
<th>Bidder</th>
<th>Valuation</th>
<th>Bid Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

\texttt{contract CryptoRomeAuction {}
\quad \text{uint256 public highestBid = 0;}
\quad \text{address payable public highestBidder;}
\quad \text{mapping(address=>uint) refunds;}
\quad \text{function bid() public payable{}
\quad \quad \text{uint duration = 1;}
\quad \quad \text{if (msg.value < (highestBid + duration))}
\quad \quad \quad \text{revert();}
\quad \quad \text{if (highestBid != 0) {}
\quad \quad \quad \text{refunds[highestBidder] += highestBid;}
\quad \quad \}
\quad \quad \text{highestBidder = msg.sender;}
\quad \quad \text{highestBid = msg.value;}
\quad \}}

C wins and pays $6$
Threats to “Smart” Auction Fairness

Untruthful behaviors

Collusion among bidders

• C wins but pays only $5
• Auctioneer loses

• B and C win and only pay $4 in total
• Both auctioneer and other bidders lose
A “Smart” Ponzi Scheme

function enter(address inviter) public {
    if ((msg.value < 1 ether) ||
        (tree[msg.sender].inviter != 0x0) ||
        (tree[inviter].inviter == 0x0)) throw;

    tree[msg.sender] = User({itself: msg.sender, inviter: inviter});

    address current = inviter;
    uint amount = msg.value;
    while (next != top) {
        amount = amount/2;
        current.send(amount);
        current = tree[current].inviter;
    }
    current.send(amount);
}
Moral of the story

Contract participants’ interpretation ≠ how the game rules are actually written

Malicious contract owner (or other participants) ≠ Benign Participants
Smart Contracts: Security vs Fairness

Security ≠ Fairness

Developer ≠ Attacker

Malicious participants ≠ Benign Participants
Establishing Trust between Contending Parties

To establish trust

Security Checker

Fairness Checker

Contract Implementation
A Typical Security Checker

• Check for pre-defined (high-profile) *attack patterns*
  • Reentrancy
    • The DAO attack (3.5 million Ether stolen, worth $45 million USD)
  • Exception Disorder
  • Gasless Send
  • Integer Overflow/underflow
    • The Proof of Weak Hand (PoWH) coin
      • 866 Ether stolen
    • ...
• Easy to miss real issues or find a lot of spurious bugs
Pattern-Based Security Checkers

Attacker Contract

```solidity
function attackDao(){
    dao.withdraw(...);
}

function() payable{
    dao.withdraw(...);
}
```

DAO Contract

```solidity
mapping(address => uint) balances;

function withdraw(uint amount){
    require(balances[msg.sender] >= amount);
    msg.sender.call.value(amount)();
    balances[msg.sender] -= amount;
}
```

- **Withdraw check -> update -> send**: ✓
- **Withdraw check -> revert**: ✗

Non-exploitable reentrancy – withdraw cannot go beyond authorization
Security checker that knows you well

• Key insights:
  • Vulnerabilities happen due to the mismatch between the externally visible balance and the internal bookkeeping
  • This applies to many types of vulnerabilities

• Two invariants to hold for all “reasonable” contracts:
  • Balance invariant (intra-contract)
  • Transaction invariant (inter-contract)
  • These include but are not limited to all ERC-20 contracts
• **Balance Invariant.** For every contract \( <a, \text{bal}, P, M> \), 
\[
\sum_{p \in P} M(p) - \text{bal} = K,
\]
where \( K \) is a constant.

• Example in contracts Attacker - DAO
  
  • before: \((10 + 15) - 25 = 0\)
  
  • after: \((10 + 20) - 30 = 0\)
**Transaction Invariant**

- **Transaction Invariant.** For every outgoing transaction \(<a, r, v>\),  
  \[ \Delta(M(r)) + \Delta(r.\, bal) = 0, \]  
  where \(\Delta(x) = post(x) - pre(x)\) and \(pre(x)\) and \(post(x)\) denote value of a variable \(x\) before and after a transaction.

- **Example in contract Attacker – DAO**
  - \(\Delta(DAO.\, M) = -5\) and \(\Delta(\text{attacker.bal}) = +5\)
Invariant Violation in DAO Attack

**Attacker Contract**

```solidity
function attackDao(){
    dao.withdraw(5);
}

function() payable{
    dao.withdraw(5);
}
```

**DAO Contract**

```solidity
mapping(address => uint) balances;

function withdraw(uint amount) {
    require(balances[msg.sender] ≥ amount);
    msg.sender.call.value(amount)();
    balances[msg.sender] -= amount;
}
```

The balance invariant is violated!
ContraMaster: Oracle-Supported Fuzzing

Security Checker

[ICSE’18]
[TDSC’20]
New Attack Surfaces

• Discovered 3 types of new attacks (not reported by other tools)
  • Incorrect access control
    • E.g., CreditDepositBank
  • Honey trap
    • E.g., ETH_VAULT and WhaleGiveaway
    • Violating transaction invariants
  • Deposit less and withdraw more
    • E.g., LZLCoin
    • Violating balance invariants
  • More details can be found at: https://sites.google.com/view/contramaster
There is no objective standard of “fairness”. “Fairness” is strictly in the eye of the beholder... To a producer or seller, a “fair” price is a high price. To the buyer or consumer, a “fair” price is a low price. How is the conflict to be adjudicated?

Define Fairness Properties

• Challenges in defining fairness
  • Fairness can be subjective
  • Fairness ≠ Equality ≠ Equity (in contrast to the “unbiased” definition)

• Consider smart contract as a game form
  • A number of players: $N = \{1, 2, \ldots, n\}$
  • An action set for each: $\Theta_1, \Theta_2, \ldots, \Theta_n$
  • An outcome function:
    • $d: \Theta \rightarrow O$ (allocation function)
    • $t: \Theta \rightarrow \mathbb{R}^n$ (transfer function)

• Preference (utility) function (individual-specific)
  • $u_i: O \rightarrow \mathbb{R}$
Focusing on generic fairness properties, i.e., independent from individual preferences.

E.g., Truthfulness

Other considered fairness properties:
- 2-collusion freeness
- Optimality
- Efficiency
- …

Automated Checker

Smart Contract
Mapping Smart Contracts into Mechanism Models

What defines the mechanism outcome?

1. Allocation function
   - Who is the winner?

2. Transfer (pricing) function
   - How much should the winner pay?

Fairness Checker
Mapping Smart Contracts into Mechanism Models

Some contract annotation can be automated: e.g., ERC-1202 (voting), ERC-1815 (blind auction)

```solidity
contract CryptoRomeAuction {
    /** FairCon Annotations */
    @individual(msg.sender, msg.value, VALUE)
    @allocate(highestBidder)
    @price(highestBid)
    @outcome(bid())
    *
    uint256 public highestBid = 0;
    address payable public highestBidder;
    mapping(address=>uint) refunds;
    function bid() public payable{
        uint duration = 1;
        if (msg.value < (highestBid + duration)){
            revert();
        } if (highestBid != 0) {
            refunds[highestBidder] += highestBid;
        }
        highestBidder = msg.sender;
        highestBid = msg.value;
    }
}
```

3-player mechanism model

CryptoRomeAuction := (msgs1, msgvalue1, −)
mişs1, msgvalue2, −)
mişs3, msgvalue3, −)

assume: (not (msgs1 < msgs1 + 1)) and
(not (msgs2 < msgs2 + 1))

allocate: argmax(msgvalue1, msgvalue2, msgvalue3)

price: max(msgvalue1, msgvalue2, msgvalue3)

Synthesizing mechanism models with symbolic execution

[FSE’20] Ye Liu, Yi Li, Shang-Wei Lin, Rong Zhao
Fairness Proof: from k-player to n-player

- **ALLOCATE** = argmax(BID)  (TopBidder)
- **PRICE** = max(BID)  (1st-Price)
- **PRICE** = max(BID\{BID[argmax[BID]]})  (2nd-Price)

Smart Contract

Find Invariants

Verifier

unverified

Veriﬁed

unknown

Fair

K-players' Model

Check Property

Counter Examples

unfair
Story 3

When the boundary between security and fairness becomes blurry …
DeFi is an ecosystem of financial applications that are built on blockchain using smart contracts.

Source: https://defipulse.com/
Source: https://thedefiant.io/defi-projects-map/
DeFi “Money Lego”

- **Composability** is one of the key features of DeFi applications
"Bounded Loss" Property Violation

"The loss of a liquidity provider is bounded by a certain value (20%) of the original deposit."

[DeFi'21] Palina Tolmach, Yi Li, Shang-Wei Lin, Yang Liu
“Bounded Loss” Property Violation

“The loss of a liquidity provider is bounded by a certain value (20%) of the original deposit.”

Overutilization

[DeFi’21] Palina Tolmach, Yi Li, Shang-Wei Lin, Yang Liu
Moral of the story

• Reality is often more complicated
  • A contract behind one game may become a player of another
  • A player may play multiple games simultaneously
  • All contracts/games can potentially be hostile

• Sometimes, fairness is security
  • There are “technical” security and “economical” security (Werner et al., 2021)
    • “A DeFi protocol is technically secure if it is not possible for an attacker to obtain a risk-free profit”
    • “A DeFi protocol is economically secure if the protocol aligns incentives among all interacting agents such that non-technical exploits are economically infeasible”

• So, how do we move forward?
  • We don’t have an answer, yet …
  • May draw some inspirations from the literature
### Some open challenges

- Scalable and precise inter-contract analysis
- Easier way to write good specifications
- Collaborative development of standards
- ...
Acknowledgements

Ye Liu

Palina Tolmach

Haijun Wang

Shang-Wei Lin

Yang Liu
In code we trust? No!

Problem: establishing trust between parties with conflicting interests

Smart Contracts
- Contract owner
- Contract user

Blockchain

Decentralization
Immutability
Anonymity
Transparency

Level of trust

Smart Contracts: Security vs Fairness

Security
- Developer
- Attacker

Fairness
- Malicious participants
- Benign Participants

Establishing Trust between Contending Parties

To establish trust

Security Checker

Fairness Checker

Contract Implementation

Moral of the story

- Reality is often more complicated
  - A contract behind one game may become a player of another
  - A player may play multiple games simultaneously
  - All contracts/games can potentially be hostile
  - Sometimes, fairness is security
    - There are "technical" security and "economical" security (Werner et al., 2021)
      - "A DeFi protocol is technically secure if it is not possible for an attacker to obtain a risk-free profit"
      - "A DeFi protocol is economically secure if the protocol aligns incentives among all interacting agents such that non-technical exploits are economically infeasible"

- So, how do we move forward?
  - We don’t have an answer, yet …
  - May draw some inspirations from the literature